Developing Interim Systems

By

Jennifer Caetta

Jet Propulsion Laboratory

“Interim Systems”have:

- Small Budget.
- Legacy infrastructure and code:
  - High risk associated with undocumented “features”.
  - Hardware or software limitations.
- Priorities focused on specific capabilities.
- Minimal time frame for initial release.

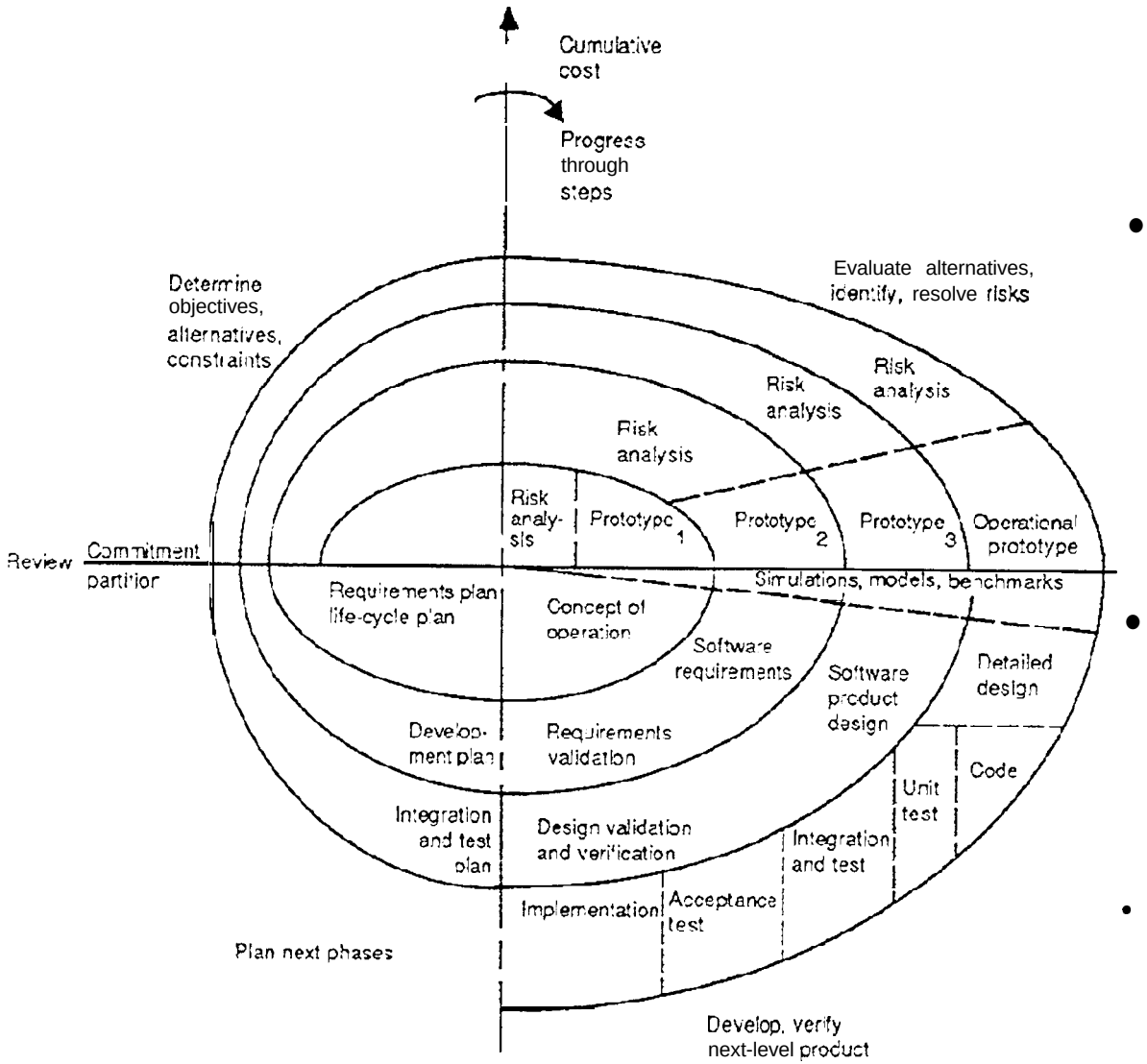
High-Level Development Issues

- Initial Approach and Design
  - Using characteristics of Interim Systems to the developer's advantage.
  - Minimizing the effects of common setbacks.
- Budgetary Concerns
- Resource Scavenging

Designing for Interim Systems: Part 1

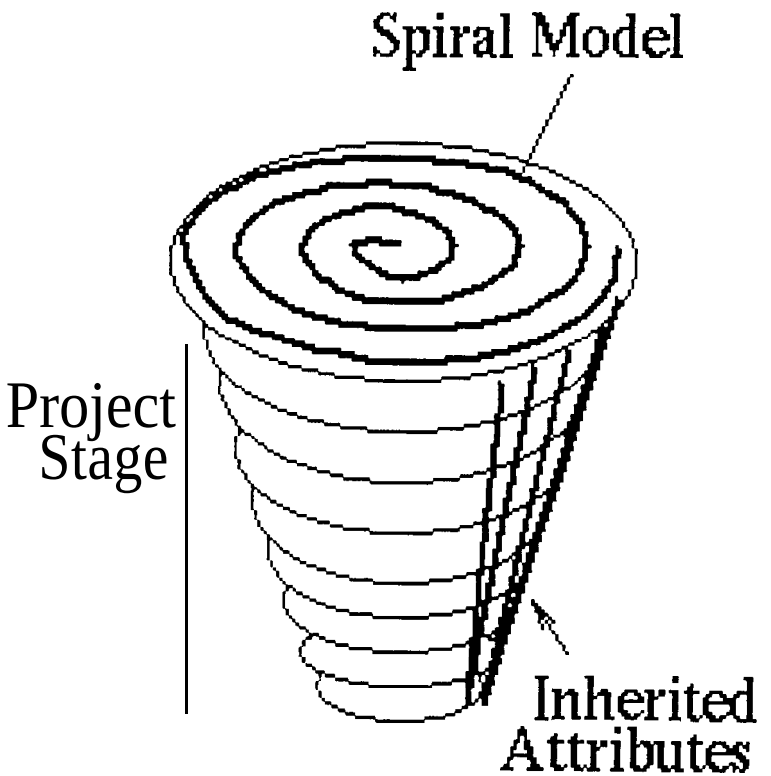
- The process model must take advantage of the unique growth and implementation patterns of interim systems.

The Spiral Software Model



- Well-developed, especially when combined with the Win-Win negotiation model.
- Is focused on a single implementation and release, and does not encompass growth.
- From "A Spiral Model of Software Development and Enhancement, B. Boehm, May 1998.

The Tornado Process Model



- Takes advantage of shared attributes of lower-level (younger) releases.
- Incorporates advantages of spiral process model.
- Organizes the growth patterns of interim systems.
- Both total and per-stage costs are easily seen.

Designing for Interim Systems: Part 2

- The design must allow quick changes to implementation-level code, due to the high risk of the legacy infrastructure.

Designing for Interim Systems: Part 3

- The developers must realize that they will be the ones re-using the software, not someone else.

Budgetary Concerns

- Usually, interim systems are for quick fixes to untenable systems which will be replaced soon.
  - No single source of funding will be willing to put much money into a “temporary” project.
  - Additionally, some funding sources demand to be the sole owner of what they buy, but can not afford to finance a robust system.

Creative Financing

- The rights-of-ownership concerns can be superseded by an awareness of the benefits of co-funding (robustness, speed, etc.)
  - Shareware-style funding: customers can use the base product, then request and finance further development.
  - "Pay now, save later" really does work.

Resource Scavenging

- Don’t write what already exists, unless you have to pay for it. If it’s worth paying for, see “Creative Financing”.
- Find who wrote the legacy software and become really good friends with them. Make them cookies or buy them beer in return for help; money usually isn’t worth it to them.

Resource Scavenging, cont.

- Find out what restrictions exist on the preferred implementation early in the design phase. Frequently, those reasons are no longer relevant.
- If restrictions cannot be lifted, design the code such that if the preferred design is ever allowed that the implementation can be “swapped in”.